**PROGRAM CODE:**

#include <sys/ipc.h>

# define NULL 0

#include <sys/shm.h>

#include <sys/types.h>

#include<unistd.h>

#include<stdio.h>

#include<stdlib.h>

#include<string.h>

#include <sys/wait.h>

#include <stdio\_ext.h>

#include<ctype.h>

void main()

{

int pid;

char \*a,\*b;

char name[50];

int id,i;

id=shmget(IPC\_PRIVATE,50,IPC\_CREAT | 00666);

pid=fork();

if(pid>0) //in parent

{

a=shmat(id,NULL,0);

printf("\nEnter a name: ");

scanf("%s",a);

wait(NULL);

shmdt(a);

}

else //in child

{

sleep(10);

b=shmat(id,NULL,0);

strcpy(name,b);

for(i=0;name[i]!='\0';i++)

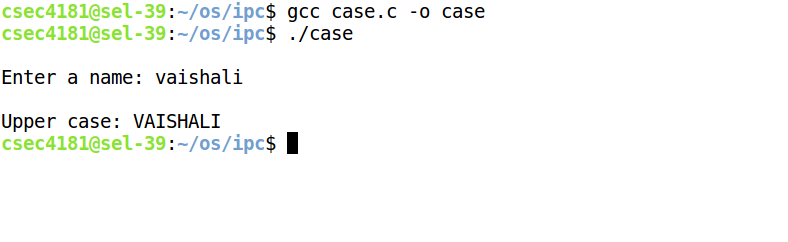
name[i]=toupper(name[i]);

printf("\nUpper case: %s\n",name);

shmdt(b);

}}

**OUTPUT:**



**PROGRAM CODE:**

**CLIENT**

#include<sys/ipc.h>

#define NULL 0

#include<sys/shm.h>

#include<sys/types.h>

#include<unistd.h>

#include<stdio.h>

void main()

{

char \*str;

int id,i;

id=shmget(555,200,IPC\_CREAT | 00666);

str=shmat(id,NULL,0);

printf("\nEnter file name: ");

scanf("%s",str);

sleep(10);

printf("\n%s\n",str);

shmdt(str);

shmctl(id,IPC\_RMID,NULL);

}

**SERVER**

#include<sys/ipc.h>

#define NULL 0

#include<sys/shm.h>

#include<sys/types.h>

#include<unistd.h>

#include<stdio.h>

#include<string.h>

#include<fcntl.h>

void main()

{

char \*str,buf[2];

int id,i,fd;

id=shmget(555,200,IPC\_CREAT | 00666);

str=shmat(id,NULL,0);

fd=open(str,O\_RDONLY);

if(fd<0)

{

printf("\nNo such file exists!\n");

strcpy(str,"Sorry,File does not exist!");

}

else

{ printf("\nFile successfully transferred\n");

read(fd,str,200);

}

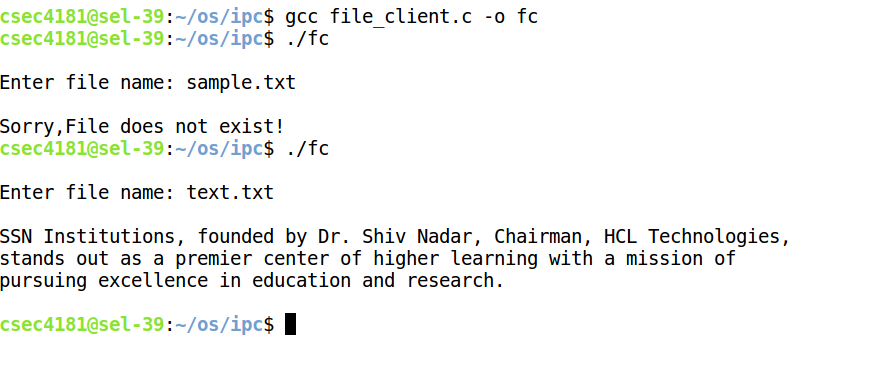
close(fd);

shmdt(str);

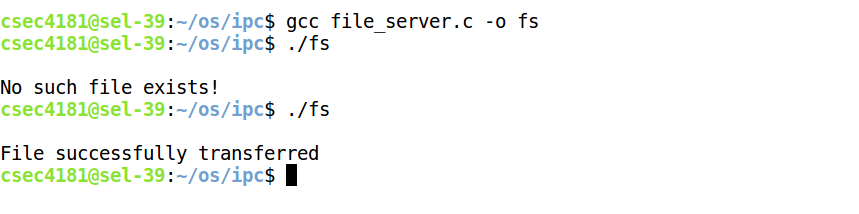
}

**OUTPUT:**

**CLIENT**

****

**SERVER**

****

**PROGRAM CODE:**

**SERVER**

#include<sys/ipc.h>

#define NULL 0

#include<sys/shm.h>

#include<sys/types.h>

#include<unistd.h>

#include<stdio.h>

#include<string.h>

void main()

{

char \* str;

char end\_str[4]="end";

int id;

id=shmget(111,100, IPC\_CREAT | 00666);

str=shmat(id,NULL,0);

printf("\nEnter 'end' to end messaging");

do{

printf("\nMessage from client: %s",str);

printf("\nEnter your message: ");

fgets(str,100,stdin);

sleep(15);

}while(strcmp(str,end\_str)!=10);

shmdt(str);

shmctl(id,IPC\_RMID,NULL);

printf("\nChat ended\n");

}

**CLIENT**

#include<sys/ipc.h>

#define NULL 0

#include<sys/shm.h>

#include<sys/types.h>

#include<unistd.h>

#include<stdio.h>

#include<string.h>

void main()

{

char \* str;

char end\_str[4]="end";

int id;

id=shmget(111,100, IPC\_CREAT | 00666);

str=shmat(id,NULL,0);

printf("\nEnter 'end' to end messaging");

do{

printf("\nEnter your message: ");

fgets(str,100,stdin);

sleep(15);

printf("\nMessage from server: %s",str);

}while(strcmp(str,end\_str)!=10);

shmdt(str);

shmctl(id,IPC\_RMID,NULL);

printf("\nChat ended\n");

}

**OUTPUT:**

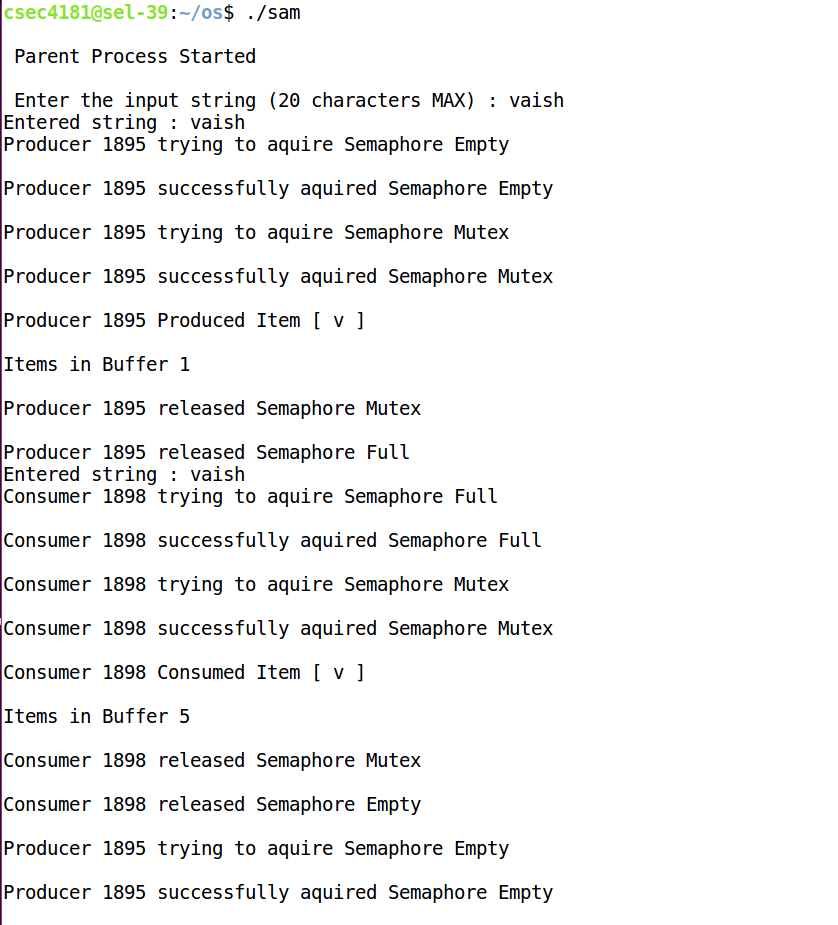
**SERVER**

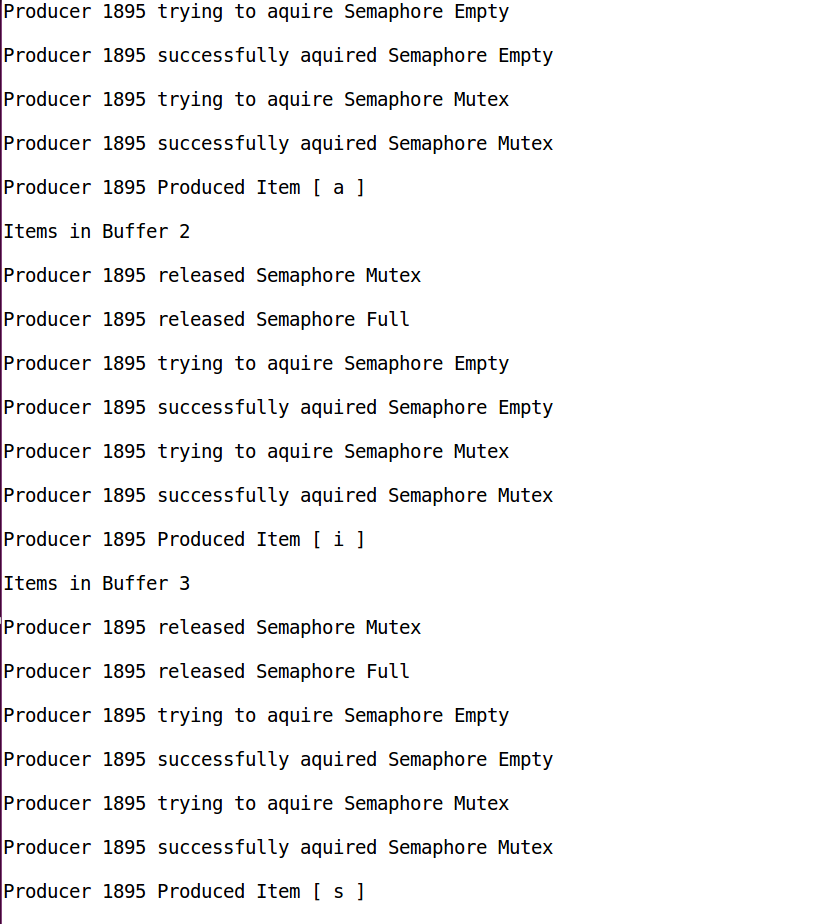
****

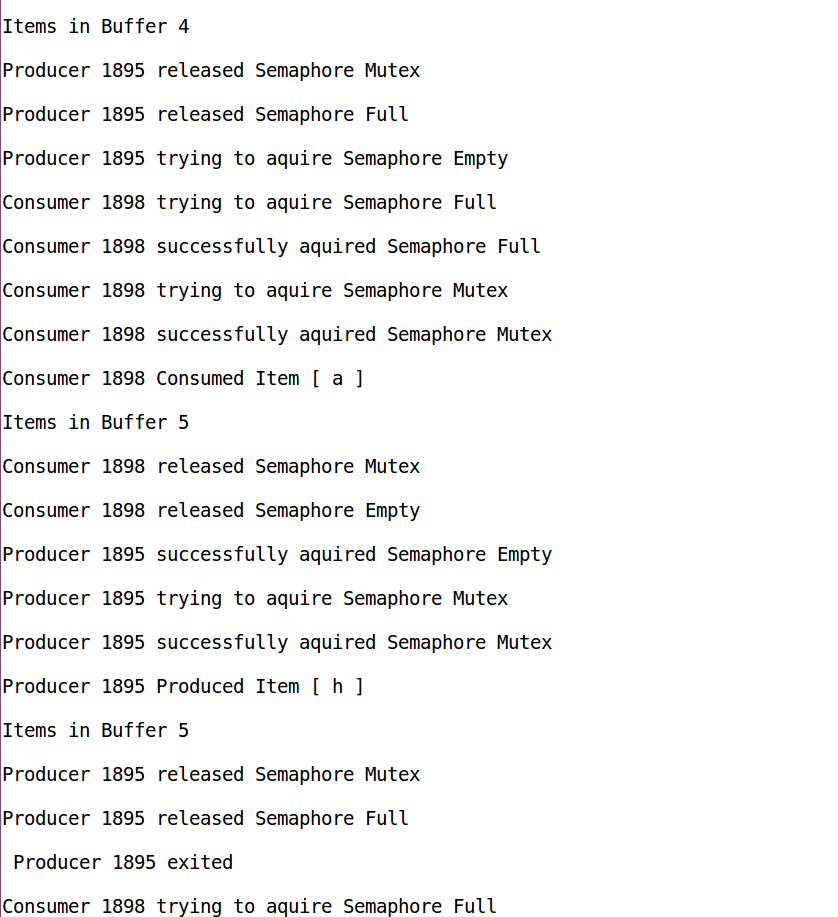
**CLIENT**

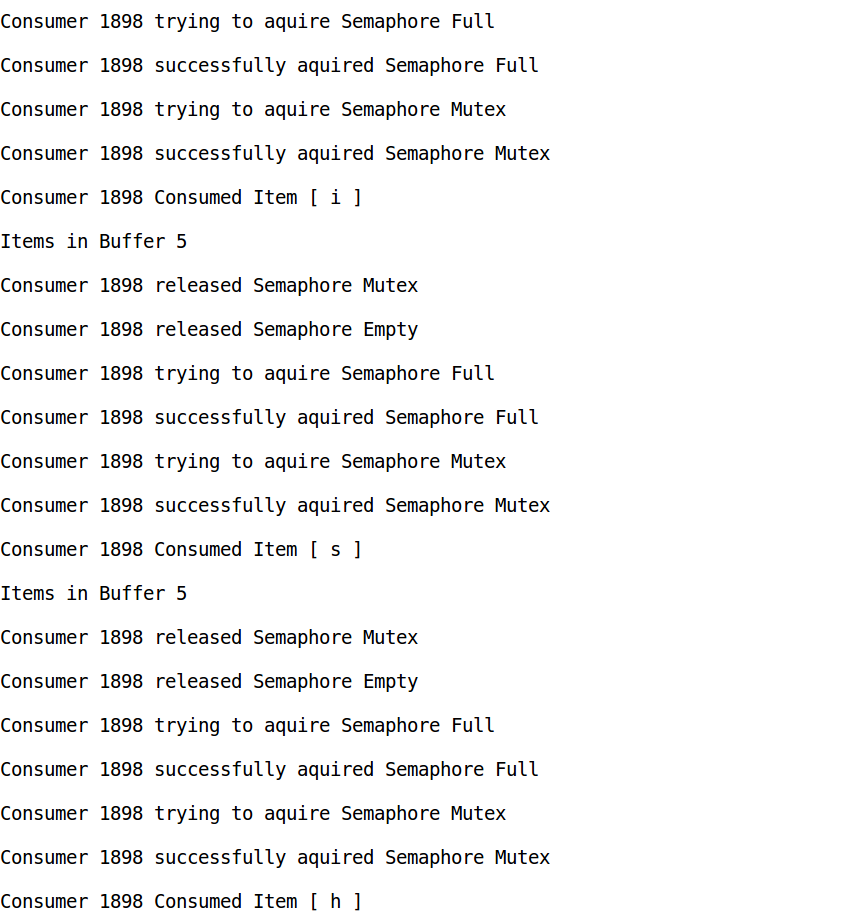
****

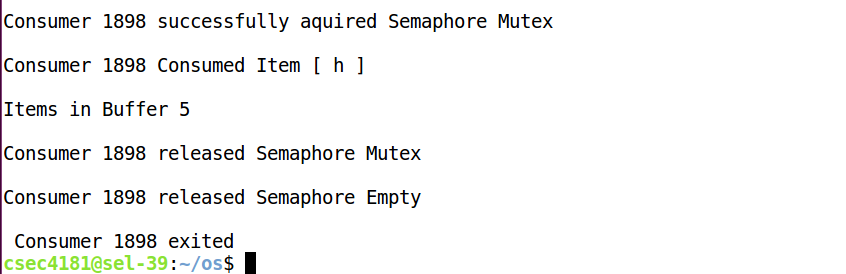
**OUTPUT:**

****

****

****

****

****

**PROGRAM CODE:**

**SERVER**

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <semaphore.h>

#include <pthread.h>

#include <sys/ipc.h>

#include <sys/shm.h>

#include <sys/sem.h>

#include <sys/errno.h>

#include <sys/types.h>

#include<sys/wait.h>

#include<unistd.h>

#include<ctype.h>

extern int errno;

#define SIZE 10 /\* size of the shared buffer\*/

#define VARSIZE 1 /\* size of shared variable=1byte\*/

#define SHMPERM 0666

int segid; /\* id for shared memory bufer \*/

int empty\_id;

int full\_id;

int mutex\_id;

char \* buff;

sem\_t \*empty;

sem\_t \*full;

sem\_t \*mutex;

int p=0;

void produce()

{

int n,d;

char t[10];

while (1)

{

if(p>=5){

printf("\n Server %d exited \n",getpid());

wait(NULL);

exit(1);

}

printf("\nServer %d trying to aquire Semaphore Empty \n",getpid());

sem\_wait(empty);

printf("Server %d successfully aquired Semaphore Empty \n",getpid());

printf("Server %d trying to aquire Semaphore Mutex \n",getpid());

sem\_wait(mutex);

printf("\nServer %d successfully aquired Semaphore Mutex \n",getpid());

n = rand() % 50 + 1;

sprintf (t,"%d",n);

strcpy(buff,t);

printf("\nServer %d sent item [ %d ] \n",getpid(),n);

p++;

sem\_post(mutex);

printf("\nServer %d released Semaphore Mutex \n",getpid());

sem\_post(full);

printf("\nServer %d released Semaphore Empty \n",getpid());

sleep(10);

}

}

int main()

{

segid = shmget(111,50,IPC\_CREAT|00666);

empty\_id=shmget(IPC\_PRIVATE,sizeof(sem\_t),IPC\_CREAT|IPC\_EXCL|SHMPERM);

full\_id=shmget(IPC\_PRIVATE,sizeof(sem\_t),IPC\_CREAT|IPC\_EXCL|SHMPERM);

mutex\_id=shmget(IPC\_PRIVATE,sizeof(sem\_t),IPC\_CREAT|IPC\_EXCL|SHMPERM);

buff = shmat( segid, NULL, 0 );

empty = shmat(empty\_id,(char \*)0,0);

full = shmat(full\_id,(char \*)0,0);

mutex = shmat(mutex\_id,(char \*)0,0);

sem\_init(empty,1,SIZE);

sem\_init(full,1,0);

sem\_init(mutex,1,1);

produce();

shmdt(buff);

shmdt(empty);

shmdt(full);

shmdt(mutex);

shmctl(segid, IPC\_RMID, NULL);

semctl( empty\_id, 0, IPC\_RMID, NULL);

semctl( full\_id, 0, IPC\_RMID, NULL);

semctl( mutex\_id, 0, IPC\_RMID, NULL);

sem\_destroy(empty);

sem\_destroy(full);

sem\_destroy(mutex);

printf("\n Server Exited \n\n");

return 0;

}

**CLIENT**

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <semaphore.h>

#include <pthread.h>

#include <sys/ipc.h>

#include <sys/shm.h>

#include <sys/sem.h>

#include <sys/errno.h>

#include <sys/types.h>

#include<sys/wait.h>

#include<unistd.h>

#include<ctype.h>

extern int errno;

#define SIZE 10 /\* size of the shared buffer\*/

#define SHMPERM 0666

int segid; /\* id for shared memory bufer \*/

int empty\_id;

int full\_id;

int mutex\_id;

char \* buff;

sem\_t \*empty;

sem\_t \*full;

sem\_t \*mutex;

int c=0;

void consume()

{

while (1)

{

if(c>=5)

{

printf("\n Client %d Exited \n",getpid());

exit(1);

}

printf("\nClient %d trying to aquire Semaphore Full \n",getpid());

sem\_wait(full);

printf("\nClient %d successfully aquired Semaphore Full \n",getpid());

printf("Client %d trying to aquire Semaphore Mutex \n",getpid());

sem\_wait(mutex);

printf("\nClient %d successfully aquired Semaphore Mutex\n",getpid());

printf("\nClient %d received item [ %s ] \n",getpid(),buff);

c++;

sem\_post(mutex);

printf("\nClient %d released Semaphore Mutex \n",getpid());

sem\_post(empty);

printf("\nClient %d released Semaphore Empty \n",getpid());

sleep(10);

}

}

int main()

{

segid = shmget(111,50,0);

empty\_id=shmget(IPC\_PRIVATE,sizeof(sem\_t),IPC\_CREAT|IPC\_EXCL|SHMPERM);

full\_id=shmget(IPC\_PRIVATE,sizeof(sem\_t),IPC\_CREAT|IPC\_EXCL|SHMPERM);

mutex\_id=shmget(IPC\_PRIVATE,sizeof(sem\_t),IPC\_CREAT|IPC\_EXCL|SHMPERM);

buff = shmat( segid, NULL, 0 );

empty = shmat(empty\_id,(char \*)0,0);

full = shmat(full\_id,(char \*)0,0);

mutex = shmat(mutex\_id,(char \*)0,0);

sem\_init(empty,1,0);

sem\_init(full,1,SIZE);

sem\_init(mutex,1,1);

consume();

shmdt(buff);

shmdt(empty);

shmdt(full);

shmdt(mutex);

shmctl(segid, IPC\_RMID, NULL);

semctl( empty\_id, 0, IPC\_RMID, NULL);

semctl( full\_id, 0, IPC\_RMID, NULL);

semctl( mutex\_id, 0, IPC\_RMID, NULL);

sem\_destroy(empty);

sem\_destroy(full);

sem\_destroy(mutex);

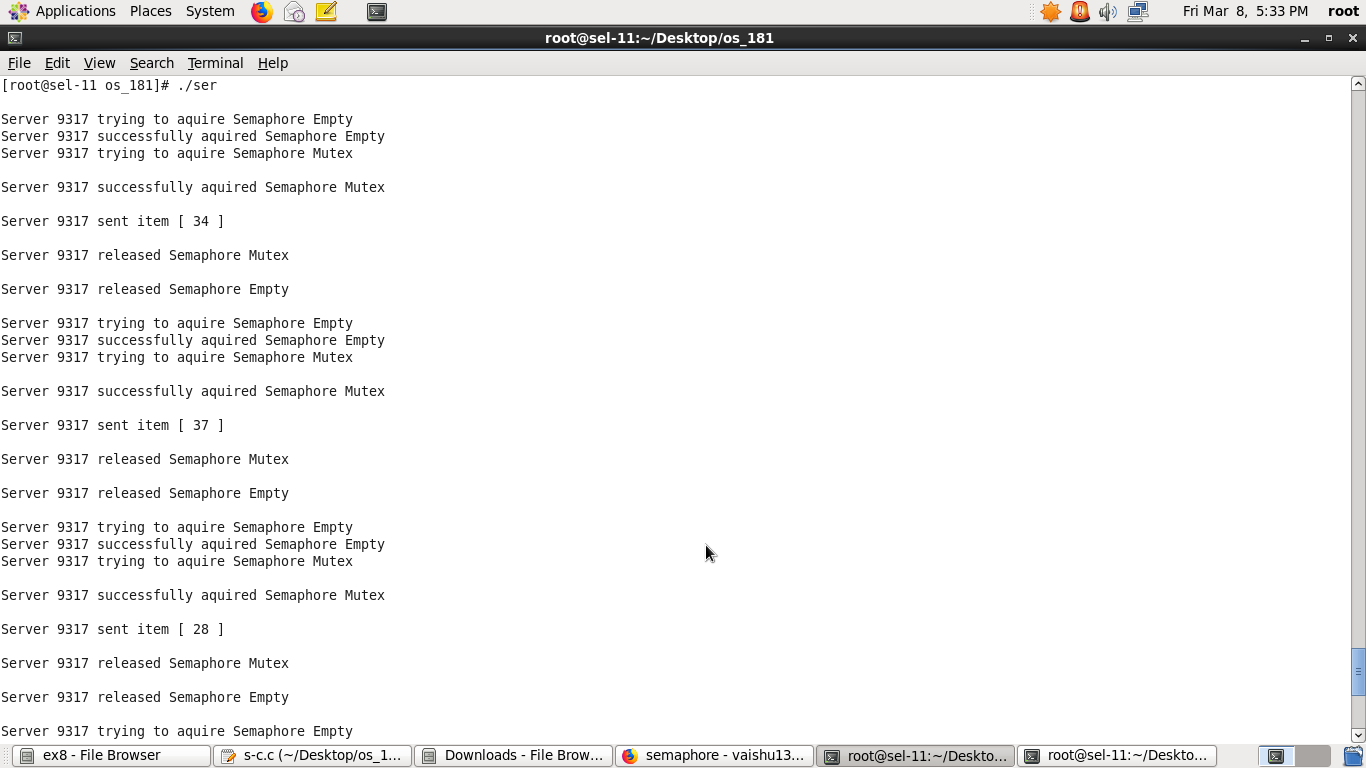
printf("\n Client Exited \n\n");

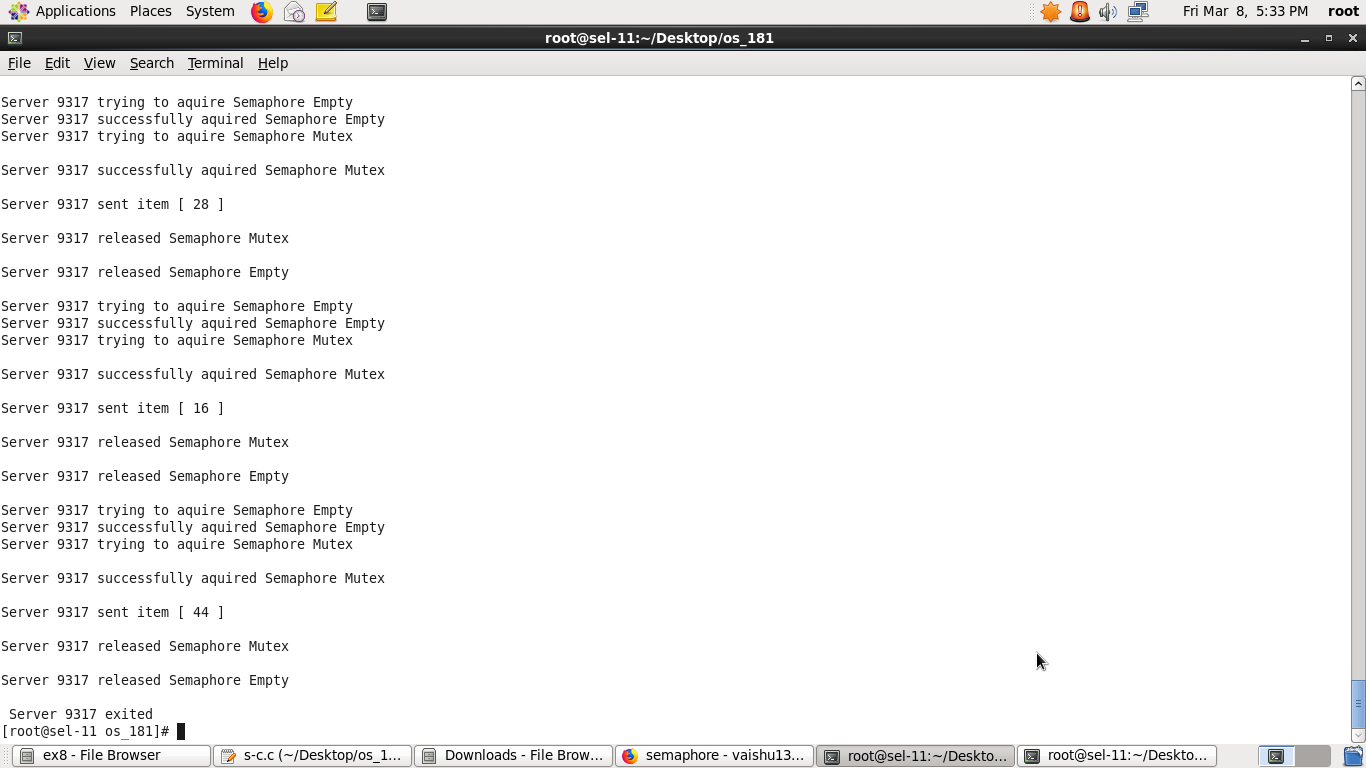
return(0);

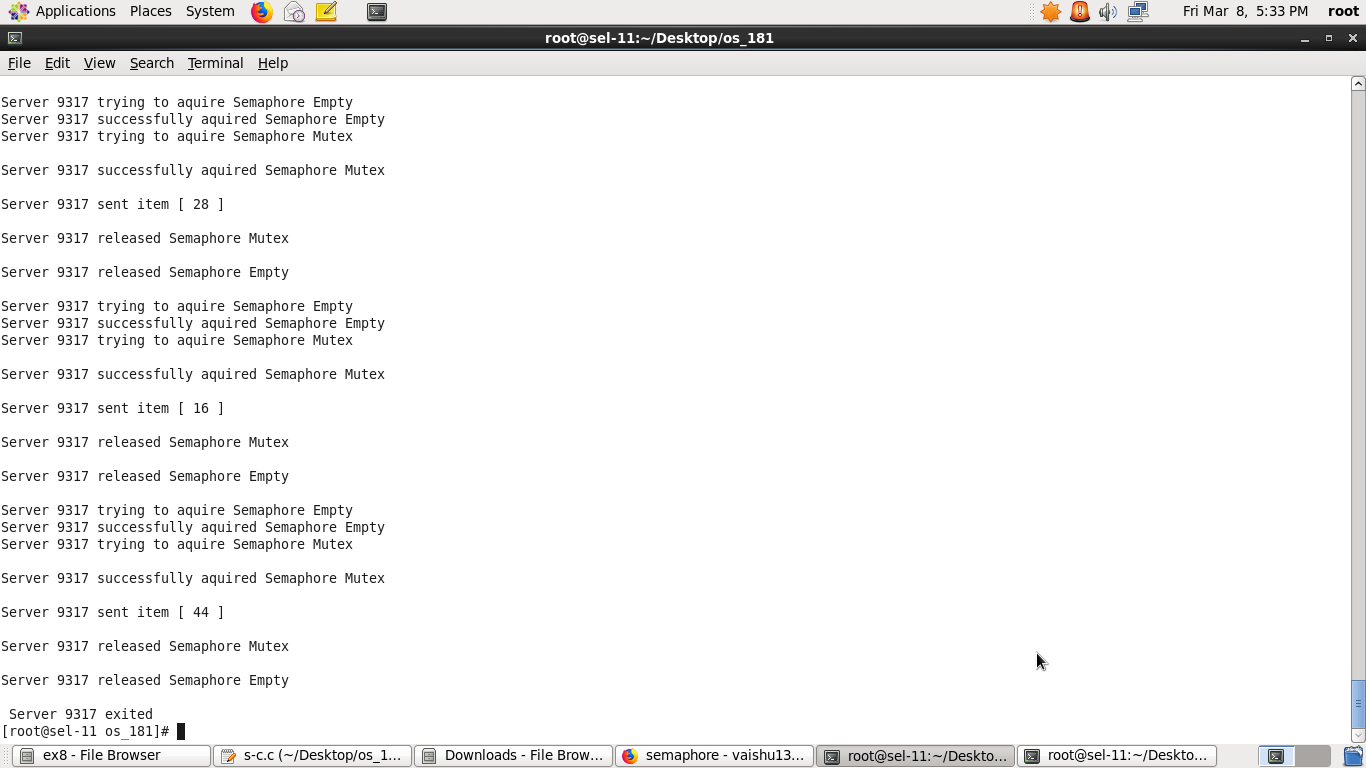
}

**OUTPUT:**

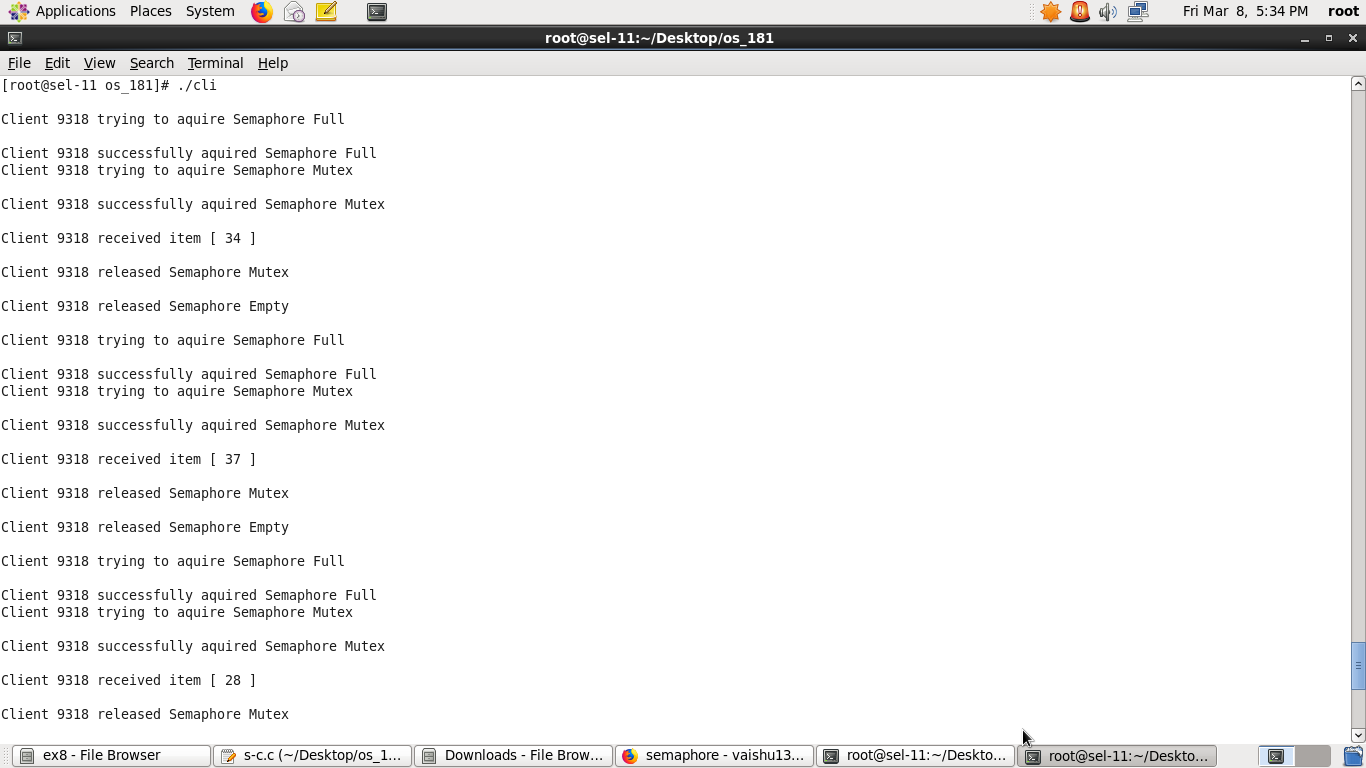
**SERVER**

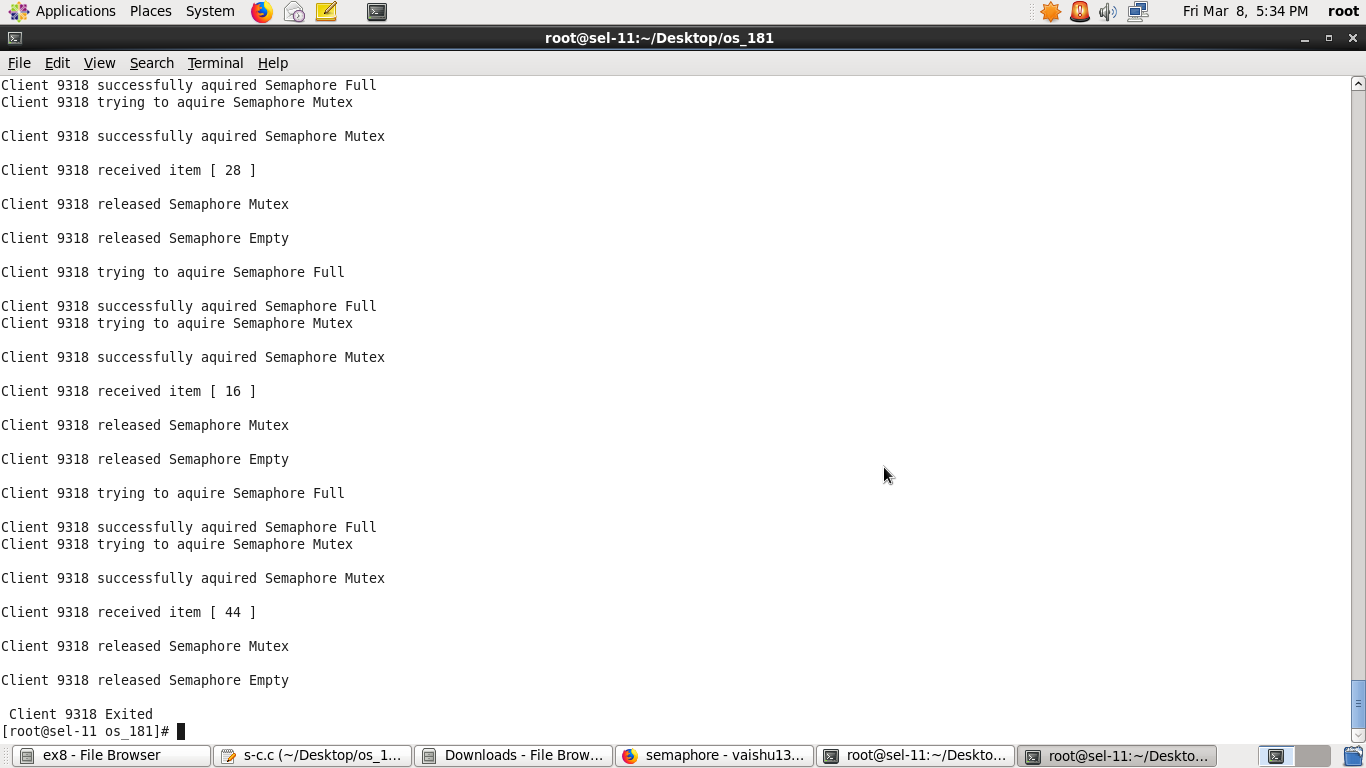
****

****

****

**CLIENT**

****

****

**PROGRAM CODE:**

#include<stdio.h>

#include<stdlib.h>

void main()

{

int p,r,flag=0,i,j,k,cnt=0,si=0,x;

printf("\t\tBANKER'S ALGORITHM\n");

printf("\nEnter the no.of processes: ");

scanf("%d",&p);

printf("\nEnter the no.of resources: ");

scanf("%d",&r);

int available[r],max[p][r],allocation[p][r] ,need[p][r],finish[p],safe\_seq[p],total\_alloc[r];

for(k=0;k<p;k++)

finish[k]=0;

for(k=0;k<r;k++)

total\_alloc[k]=0;

printf("\nEnter the available no.of instances for each resource: \n");

for(i=0;i<r;i++)

{

printf("Resource %c: ",i+65);

scanf("%d",&available[i]);

}

printf("\nEnter the maximum requirement for each process: ");

for(i=0;i<p;i++)

{

printf("\nFor P%d: \n",i+1);

for(j=0;j<r;j++)

{

printf("Resource %c: ",j+65);

scanf("%d",&max[i][j]);

}

}

printf("\nEnter the allocated no.of resources for each process: ");

for(i=0;i<p;i++)

{

printf("\nFor P%d: \n",i+1);

for(j=0;j<r;j++)

{

printf("Resource %c: ",j+65);

scanf("%d",&allocation[i][j]);

total\_alloc[j]+=allocation[i][j];

}

}

printf("\nNeed for each process\n ");

for(k=0;k<r;k++) printf("%c ",k+65);

printf("\n");

for(i=0;i<p;i++)

{

printf("P%d ",i+1);

for(j=0;j<r;j++)

{

need[i][j]=max[i][j]-allocation[i][j];

printf("%d ",need[i][j]);

}

printf("\n");

}

for(j=0;j<r;j++)

available[j]=available[j]-total\_alloc[j];

i=0;

while(cnt!=p)

{

if (finish[i]==0)

{

flag=0;

for(j=0;j<r;j++)

if(need[i][j]>available[j])

flag=1;

if(flag==0)

{

for(j=0;j<r;j++)

available[j]+=allocation[i][j];

finish[i]=1;

safe\_seq[si]=i; si++;

cnt++;

}

} //end if finish

i=(i+1)%p;

} //end while

printf("\n\nSafe Sequence: ");

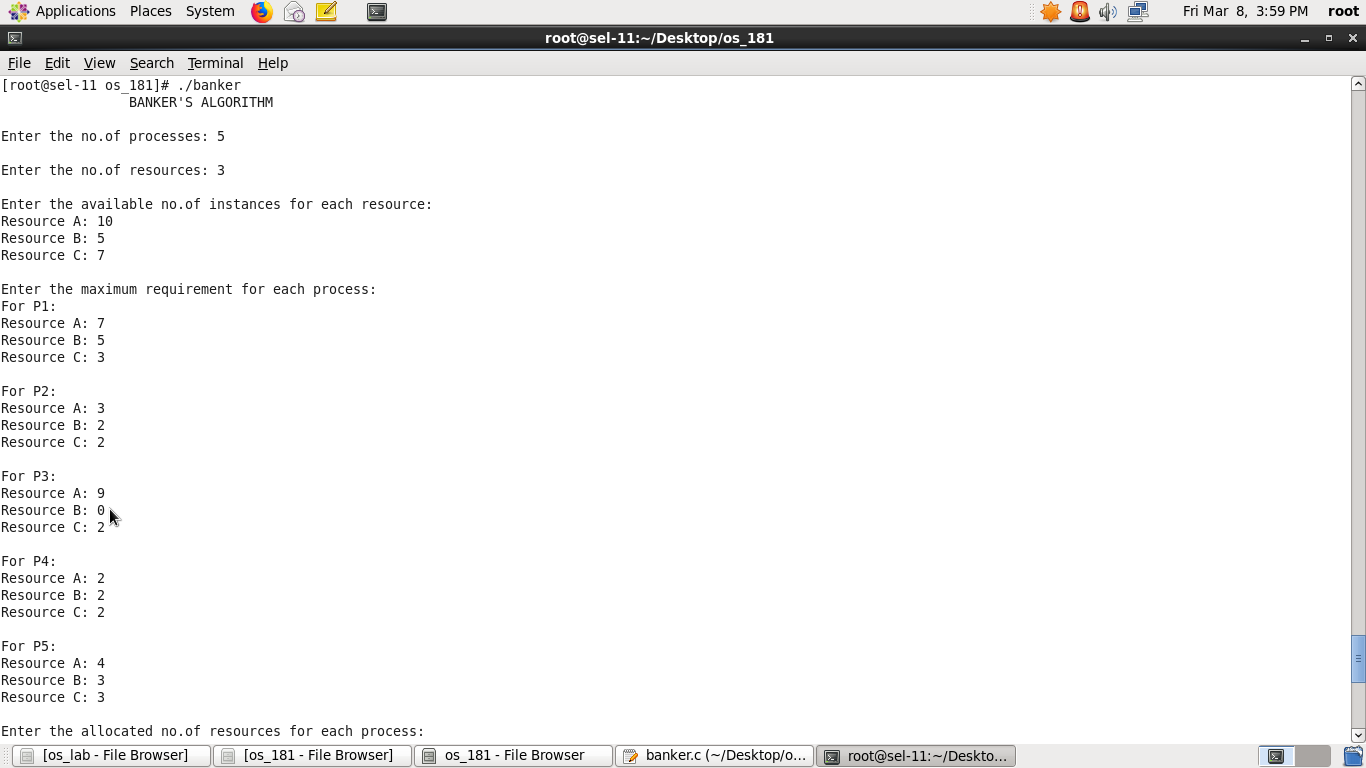
for(k=0;k<si;k++)

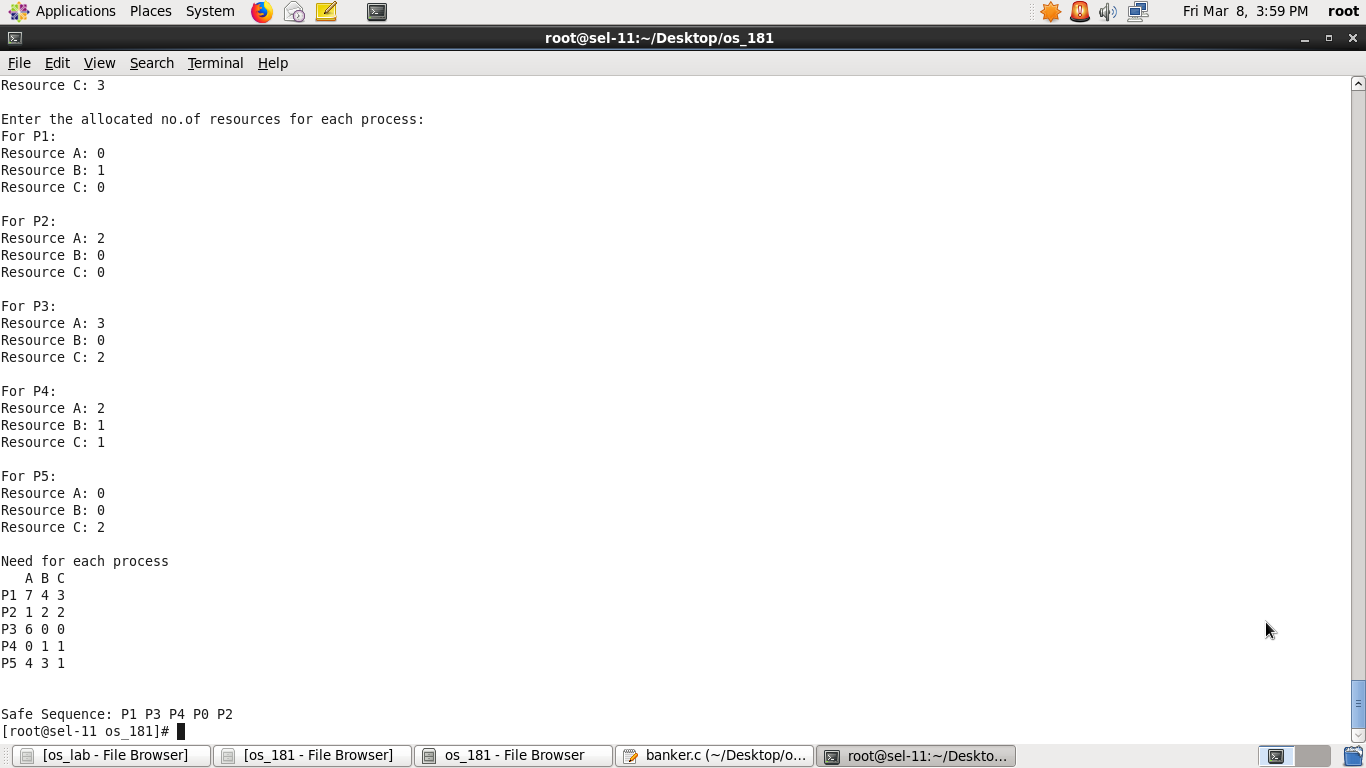
printf("P%d ",safe\_seq[k]);

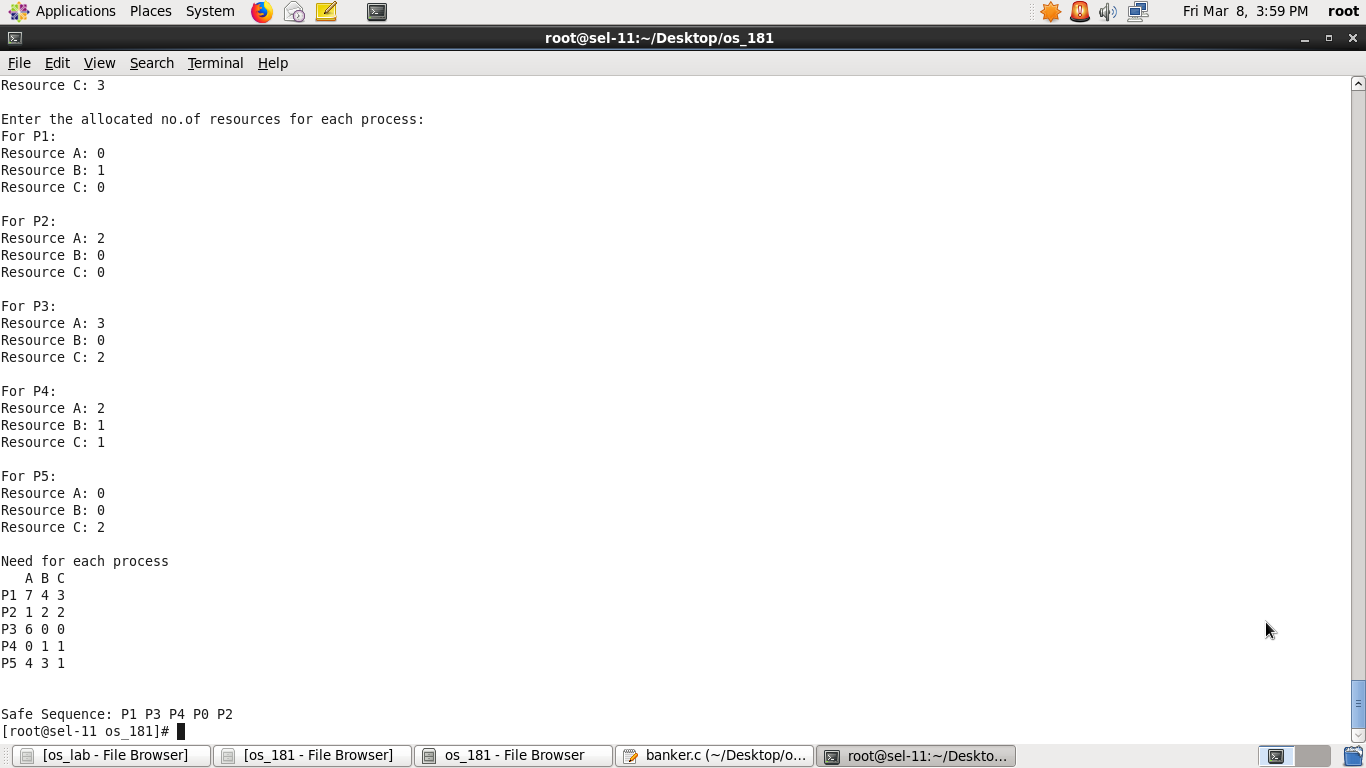
printf("\n");

}

**OUTPUT:**

****

****

****